CMPSC-265
Data Structures and Algorithms

Zaihan Yang
zyang13@suffolk.edu

Department of Math and Computer Science
Suffolk University

Fall 2019

Recap

- Elementary sorting algorithms

Algorithm	Worst case	Average case	Best case	In-place?
Selection	$O(N^2)$	$O(N^2)$	$O(N^2)$	Yes
Insertion	$O(N^2)$	$O(N^2)$	$O(N)$	Yes
Bubble	$O(N^2)$	$O(N^2)$	$O(N)$	Yes

Comparison of Basic Sorting Algorithms

- The average-case discussed algorithms are all in O(n^2).
- In practice, insertion sort performs better than other elementary sorting algorithms.
- Sorting algorithms visualizer:

https://visualgo.net/bn/sorting

Learning Topics

- Implementing Comparable and Comparator interface to sort on objects
- The Stack data structure
- Implementation of Stack
- Applications on Stack

Sorting Objects in Java

- How to sort a list of Movie objects in terms of:
  - Their ratings
  - Their names
  - Their year
- We can implement the Comparable interface, and override the compareTo() method
- We can implement the Comparator interface, and override the compare() method.

Interface

- An interface is a reference type in Java. It is similar to class. It is a collection of abstract methods. A class implements an interface, thereby inheriting the abstract methods of the interface.

Comparable Interface

- A comparable object is capable of comparing itself with another object. The class itself must implements the java.lang.Comparable interface to compare its instances.
- Once it is implemented, this will be the by-default sorting criteria for this object.
- Check to see the codes in Movie.java

Comparator Interface

- If we want to make Movie comparable by other criteria, we can implement the Comparator interface.
- Unlike Comparable, Comparator is external to the element type we are comparing. It’s a separate class. We create multiple separate classes (that implement Comparator) to compare by different members.
- Collections class has a second sort() method and it takes Comparator. The sort() method invokes the compare() to sort objects.

Comparator Interface

To compare movies by Rating, we need to do 3 things :

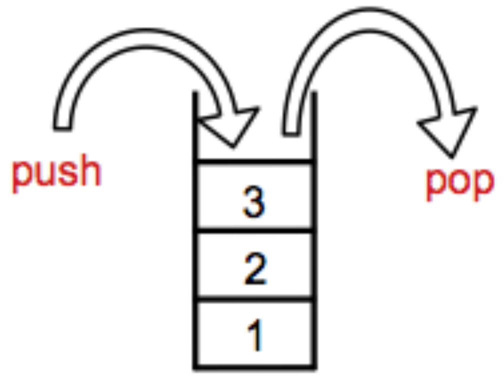
- Create a class that implements Comparator (and thus the compare() method that does the work previously done by compareTo()).
- Make an instance of the Comparator class.
- Call the overloaded sort() method, giving it both the list and the instance of the class that implements Comparator.
- Check the codes for Movie.java

Comparable vs. Comparator Interface

- Comparable is meant for objects with natural ordering which means the object itself must know how it is to be ordered. For example Roll Numbers of students. Whereas, Comparator interface sorting is done through a separate class.
- Logically, Comparable interface compares “this” reference with the object specified and Comparator in Java compares two different class objects provided.
- If any class implements Comparable interface in Java then collection of that object either List or Array can be sorted automatically by using Collections.sort() or Arrays.sort() method and objects will be sorted based on there natural order defined by CompareTo method.

Stacks

- Abstract data structure
  - Implementation is hidden
- LIFO (Last In First Out) or FILO (First in Last Out)
- Only the last element (top of the stack) can be accessed
- When the top element is removed, we can access the next element,...
- The first in element will be the last one out ; The last one in will be the first one out.

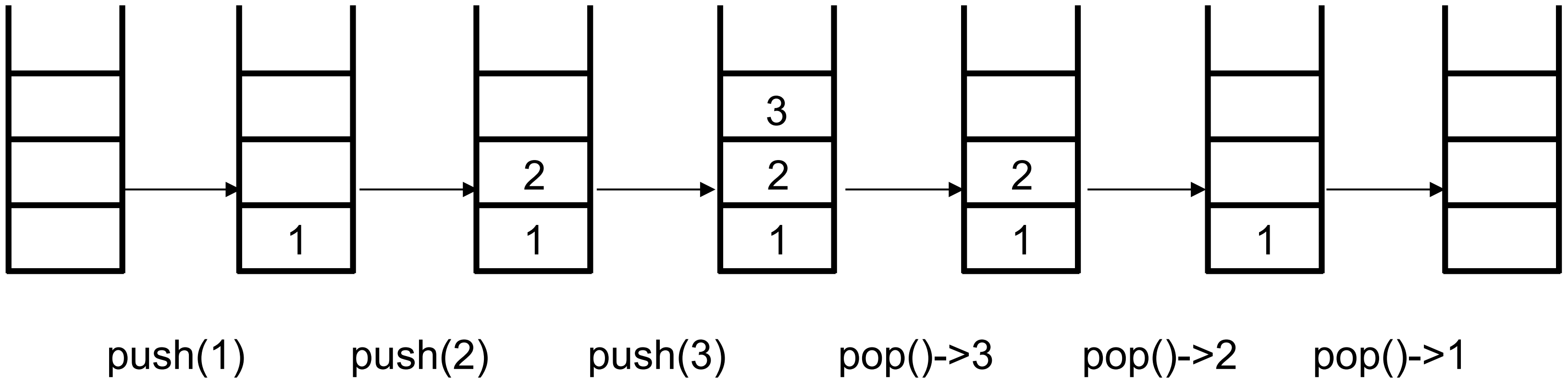


Stack

- Operations we consider on a Stack
  - Adding new element: push(a)
  - Delete an element: pop() // cannot only access the top one, and remove it from the stack.
  - Checking on the top one: peek() // just check the top one without removing it.
  - Determine whether the stack is empty: isEmpty()

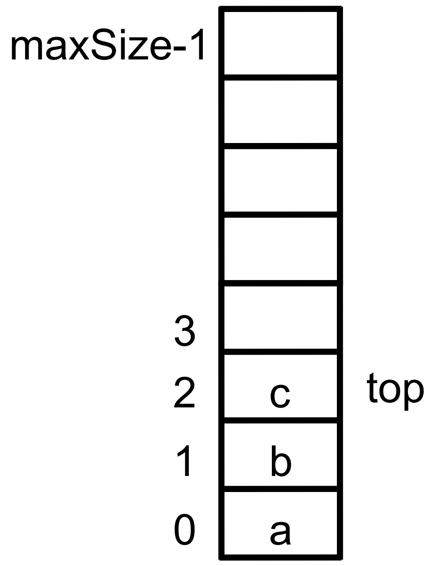
Stack Example

- Push and pop: can reverse the order.



Stack Implementation

- Can use an array to hold elements
- Methods needed:
  - pop()
  - push(element)
  - peek()
  - isEmpty()
  - isFull()
  - size()
- Fields needed:
  - maxSize (capacity of the stack)
  - top (holds the index of the top element)
  - stackArray (array of items in stack)



Stack Class

```
class StackX
   {
   private int maxSize;          // size of stack array
   private long[] stackArray;
   private int top;              // top of stack
//-----------------------------------------------------------------
   public StackX(int s)           // constructor
      {
      maxSize = s;               // set array size
      stackArray = new long[maxSize];   // create array
      top = -1;                  // no items yet
      }
```

Stack Class

```
//--------------------------------------------------------------
   public void push(long j)    // put item on top of stack
      {
      stackArray[++top] = j;     // increment top, insert item
      }
//--------------------------------------------------------------
   public long pop()           // take item from top of stack
      {
      return stackArray[top--];  // access item, decrement top
      }
//--------------------------------------------------------------
   public long peek()          // peek at top of stack
      {
      return stackArray[top];
      }
//--------------------------------------------------------------
   public boolean isEmpty()    // true if stack is empty
      {
      return (top == -1);
      }
//--------------------------------------------------------------
   public boolean isFull()     // true if stack is full
      {
      return (top == maxSize-1);
      }
//--------------------------------------------------------------
   }  // end class StackX
```

Robert Lafore. 2002. Data Structures and Algorithms in Java (2 ed.). Sams, Indianapolis, IN, USA. Page 120

Stack Demo Class

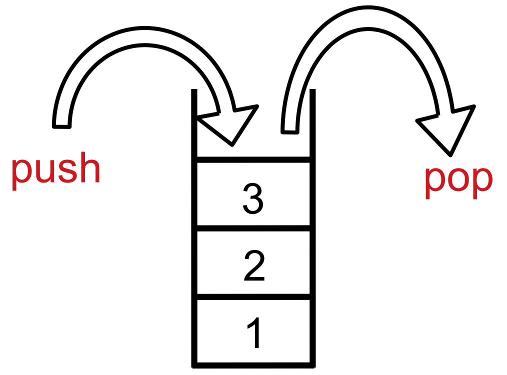
```
class StackDemo
{
        public static void main(String[] args)
        {
                StackX myStack = new StackX(5);
                myStack.push(1);
                myStack.push(2);
                myStack.push(3);

                System.out.print(myStack.size());

                while( ! myStack.isEmpty() )
                {
                        System.out.print(myStack.pop());
                }
        }
}
```

Stacks

- Very simple and very useful in many situations
  - Reverse the order
  - Undo (ctrl+z)
  - Check/parse expressions
  - Function calls



Example-Delimiter Matching

- Suppose you will read in a String representing an arithmetic expression:
- Tell if an input string has a matching parentheses
  - ((a+b)+c): correct
  - (a+b)+c): incorrect
- Use a stack
  - push ‘(‘ into stack, pop when you see ‘)’, decide based on stack empty condition.

Example-Delimiter Matching

```
public boolean checkParenthesisMatch (String exp) {
        StackX st=new StackX (exp.length());
        for(int i=0;i<exp.length();i++) {
                char ch = exp.charAt(i);
                if (ch == '(' )
                        st.push(ch);
                if (ch == ')' ) {
                        if (st.isEmpty())
                                return false;
                        else
                                st.pop();
                }
        }
        return st.isEmpty();
}
```

What if I want to check brackets [], at the same time?

What is the time complexity?

Example-Delimiter Matching

```
import java.util.Stack;
public boolean checkParenthesisMatch (String exp) {
        Stack<Character> st = new Stack<Character>();
        for(int i=0;i<exp.length();i++) {
                char ch = exp.charAt(i);
                if (ch == '(' )
                        st.push(ch);
                if (ch == ')' ) {
                        if (st.isEmpty())
                                return false;
                        else
                                st.pop();
                }
        }
        return st.isEmpty();
        }
```